REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	5 Sep 97	

4. TITLE AND SUBTITLE
A COMPUTERIZED APPROACH TO A MULTIVARIABLE, CONSTRAINED, NONLINEAR OPTIMIZATION BLENDING PROBLEM USING A MONTE CARLO SIMULATION

5. FUNDING NUMBERS

6. AUTHOR(S)
Michael A. Martinez

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Auburn University
Auburn, Alabama

8. PERFORMING ORGANIZATION REPORT NUMBER
97-116

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
THE DEPARTMENT OF THE AIR FORCE
AFIT/CIA
2950 P STREET
WPAFB OH 45433-7765

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A
Approved for public release
Distribution Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

14. SUBJECT TERMS

15. NUMBER OF PAGES
19

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

A COMPUTERIZED APPROACH TO A MULTIVARIABLE, CONSTRAINED, NONLINEAR OPTIMIZATION BLENDING PROBLEM USING A MONTE CARLO SIMULATION

Michael A. Martinez

A Design Project

Submitted to the

Industrial and Systems Engineering Faculty

of Auburn University in

Partial Fulfillment of the

Requirements for the

Master's Degree of

Industrial and Systems Engineering

Auburn, Alabama

July 23, 1997

TABLE OF CONTENTS

ABSTRACT

This design project dealt with a blending problem in the format of a multivariable, constrained, nonlinear optimization problem, where a typical application of this problem may be an incinerator that burns waste. The incinerator accepts a blend of several sources, each of which has particular characteristics in terms of thermal output. Should the different sources be inappropriately blended, it could result in a safety hazard for the entire incinerator. The problem is to determine the appropriate proportions so that the maximum value of thermal output for the blend is minimized. The maximum value will be approximated by the mean of the blend plus a selected number standard deviations. The first phase of the project was a brief study of similar blending problems by searching relevant literature. It was found that very little has been done on this problem in terms of computer implementation for industry.

Secondly, various possible methods for solving the blending problem were considered. Among these methods were classical calculus, gradient searches, exhaustive searches, and iterative direct searching. An extension of an iterative direct search is a Monte Carlo simulation, where a statistical approach is taken. The Monte Carlo search method is based on the theory that the optimization problem has a distribution of answers and that a random sample of those answers will yield an answer in the lower tail to within a certain degree of "accuracy." Then, the feasible region of answers will be halved and another random sample of possible answers will be taken. The search proceeds in this manner, taking random samples from an ever-decreasing region of feasible answers. The method focuses on the optimal answer.

The third aspect of the project was implementing the Monte Carlo method in computer code. The program was written in *Visual Basic 4.0 for Windows* in order to achieve user-friendliness and a graphical user interface. The program contains both an optimization function and a "What-if" analysis. A "User's Manual" was also documented to assist the user.

M.I.S.E. Design Project Proposal

Essentially, my design project will consist of a blending problem. A typical application of this problem to the industrial domain would be an incinerator that burns waste. The incinerator accepts material from several sources, each of which has particular characteristics in terms of thermal output. The thermal output of each source is assumed to be normally distributed, having its own mean and standard deviation. The sources will be blended to create the final burning material. Should the different sources be inappropriately blended, it could result in a safety hazard for the entire incinerator. The problem is to determine the appropriate proportions so that the maximum value of thermal output for the blend is minimized. The maximum value will be approximated by the mean of the blend added to a selected number standard deviations. For example, an addition of three standard deviations results in a 0.00135 probability that the maximum value will be exceeded. As the number of sources increases, the problem quickly becomes more complex. The goal of the project is to produce a working model computer program that will allow user interface so that it will calculate the appropriate proportions of each source to produce an optimally safe blend for the incinerator. Additionally, the program will allow the user to play "what-if" games in order to test different blending situations. This feature will be advantageous when the blending has constraints.

The first phase of my project will be a brief study of similar blending problems. In talking with my advisor, Dr. Hool, it was discovered that very little has been done on this problem in terms of computer implementation for industry. This phase is an attempt to determine what is currently available in this field.

Secondly, a mathematical analysis will be done on the problem to determine the best solutions to the problems, as the number of sources increases. An initial hypothesis is that search methods for optimization will need to be utilized since a closed-form solution is impossible and an exhaustive computer search will be impractical. Additionally, the general program requirements and options will be determined.

The third aspect of the project will be implementing the working solution in computer code. The initial proposal is to write the program in Visual Basic in order to achieve user-friendliness and graphical user interface. After the program is flowcharted, the coding will be done. Naturally, testing and de-bugging will also be required. Optionally, a short "User's Guide" will be documented.

Finally, the write-up will be completed in order to ensure thorough documentation. As of now, the schedule for completion date is June 10, 1997. This will allow almost three months for the review, oral examination, processing of results, and payment of the graduation fee well before the deadline two-weeks prior to graduation.

PROBLEM ORIGIN

This problem originated from a real-world problem involving hazardous waste disposal through burning of hazardous waste material in an incinerator. Various sources of hazardous waste material are inputs to the incinerator, and the sources are blended into a mixture prior to insertion into the incinerator. Each source contains a distribution of Btu (British thermal unit) content (i.e. the Btu content is not a single value for a source). The incinerator cannot be safely operated if the maximum Btu of a blended input exceeds 800 Btu (the "safe limit"). The problem is to determine how to blend the sources in order to avoid burning material that has a maximum Btu value exceeding the "safe limit" of the incinerator. Specifically, the problem is to determine the proportion of each source to put into a blend so that there is a small probability (e.g. 0.001) that the maximum Btu will exceed the "safe limit."

The blend can be modeled as a linear combination of source Btu's. Each source has its own distribution of Btu content. The mean Btu of source i, $i = 1, 2, \ldots, n$, is μ_i, and the variance of Btu content of source i is σ_i^2. If the proportion of source i that is blended into the final mix is k_i, then the Btu content of the blend is a linear combination, B, of the sources. If X_i is the Btu content of source i, where X_i is a random variable, then

$$B = \sum_i k_i X_i$$

and B has a mean

$$\mu_B = \sum_i k_i \mu_i$$

and a variance

$$\sigma_B^2 = \sum_i k_i^2 \sigma_i^2$$

if the sources are independent. The independence assumption is easily met in the problem. If the Btu content of each source is normally distributed, then B is also normally distributed. If not all of the sources have normally distributed Btu's, then B may still be approximately normally distributed if enough sources are blended (as a result of the Central Limit Theorem).

If B can be assumed to be at least approximately normally distributed, then for any arbitrarily selected probability, p, where p will be small, a corresponding "maximum" value for B, defined as M_p , can be written as

$$M_p = \sum_i k_i \mu_i + Z_p \sqrt{\sum_i k_i^2 \sigma_i^2}$$

where Z_p = standardized normal distribution value corresponding to the probability p.

M_p is the (100)pth percentile of B, and is the value of B beyond which only (100)p% of the blend's Btu's will lie.

If the "safe limit" of the incinerator is S, then the blending problem becomes one of selecting the k_i such that

$$M_p \leq S$$

subject to the constraints

$$\sum_{i=1}^{n} k_i = 1$$

$$0 \leq k_i \leq b_i \quad \forall i = 1, \cdots, n$$

where $b_i = 1$ are default values.

In particular, the "ultimate" solution of the blending problem requires determination of $\min(M_p) = M_p^*$. If $M_p^* \leq S$, then the blend satisfies the incinerator's "safe limit." Of course, there are two additional situations that might occur, namely

1. $M_p^* > S$, in which case a blend cannot be created which may be "safely" processed by the incinerator.
2. There are multiple M_p (corresponding to various k_i combinations) for which $M_p \leq S$.

The blending problem specific to this project, and any blending problem of similar character, is a multivariable constrained nonlinear optimization problem. The decision variables are the k_i, and the term

$$\sqrt{\sum_i k_i^2 \sigma_i^2}$$

creates the nonlinearity of the objective function M_p. Further, several variations on the problem can be created such as maximizing a minimum value of B such that the minimum value exceeds a lower "safe limit."

PROBLEM STATEMENT

The objective of this study was to select an appropriate method to solve the blending problem, and then produce a working-model computer program to execute the selected method for solving the problem. The computer program was written in *Visual Basic 4.0 for Windows* and performs two functions, namely

1. Determines the optimal k_i (called k_i^*) to minimize M_p. The program can determine if the blend meets the "safe limit" ($M_p^* \leq S$).
2. Allows the user to conduct "what-if" analyses for user-supplied k_i values in order to determine if the resulting blend will satisfy $M_p \leq S$.

Because the number of sources will be at least two and perhaps as many as ten, the resulting problem is the following multivariable nonlinear programming problem:

$$\min\left[\sum_{i=1}^{n} k_i \mu_i + Z_p \sqrt{\sum_{i=1}^{n} k_i^2 \sigma_i^2}\right]$$

subject to

$$\sum_{i=1}^{n} k_i = 1$$

$$0 \leq k_i \leq b_i \quad \forall i = 1, \cdots, n$$

LITERATURE SEARCH

In order to evaluate what is currently available concerning blending problems, a search of relevant literature was conducted. The problem being investigated is a multivariable constrained nonlinear optimization problem. Because the problem is

nonlinear, linear programming is not a viable option. Most nonlinear optimization problems concern economics or inventory problems. The most preferred method of solving nonlinear optimization problems is by using a gradient search procedure, which employs the partial derivatives of the objective function. Optimization methods will be more fully discussed in the next section.

Nicholson (1) comments that in the industrial world a gap has been formed between the people who develop optimization methods and those who use them. Nicholson's premise is that the most accurate optimization methods will be useless unless they are, if not theoretically comprehended, at least pragmatically understood by the practitioner. An objective of this project was to implement a powerful and perhaps complex optimization method and make it useful so that a layman can easily solve blending problems.

In examining Nicholson's second volume of applications, along with other industrial mixing books, it was found that most mixing problems deal with complex mixing formulas, and tend toward chemistry problems. Virtually no material was found relating directly to solving blending problems.

A helpful resource that was discovered was Experiments with Mixtures by Cornell. However, Cornell's approach toward the mixing problem adopts design of experiments and response surface methodology. The purpose of designing a complex experiment is to construct an equation that adequately describes the response surface (8). Once the response surface is estimated, it is then optimized by selecting the appropriate levels of the factors. In this project, the response surface is assumed to be known exactly and the real problem lies in selecting the levels of the factors (in this case, k_i).

POSSIBLE SOLUTION ALGORITHMS

Multivariable constrained nonlinear optimization problems can be solved using various methods. Several methods that were considered are briefly described below. Note that use of a linear approximation of a nonlinear objective function is less accurate and was thus not considered.

1. Analytical (Calculus) Method

One method to solve a nonlinear optimization problem is the classical calculus method. This involves, in a multivariable problem, determining the first partial derivative of the objective function (M_p in this case) with respect to each variable (i.e., k_i). Since the constraint $\Sigma k_i = 1$ is used in the blending problem, only (n-1) partial derivatives must be determined. Each partial derivative must be set equal to zero, resulting in a system of equations, and the k_i then solved for to produce a global or local maximum, minimum, or a saddle point solution. This is not easy in the general case, and in particular for the blending problem. For the blending problem each partial derivative is a nonlinear function, and solutions for the k_i could require a difficult search process. A general formula for the (n-1) first partial derivatives is shown in Appendix 1, and an example for $n = 2$ sources is shown in Appendix 2. Since solution via the classical calculus method is complex, this method can seldom be used satisfactorily on even unconstrained problems (Nicholson 69). In addition, determining whether extreme points are local minima, maxima, or saddle points can be difficult.

2. Gradient Search

When the system of equations obtained by setting the partial derivatives equal to zero cannot be solved analytically, then a numerical search procedure should be used (Hillier 514-5). Using the objective function, M_p, the goal is to reach a point eventually where all of its partial derivatives are (essentially) zero (Hillier 515). The gradient, a vector of the (n-1) partial derivatives, can be calculated and used to determine the direction of the search. However, a gradient search procedure itself must employ another search procedure to determine how far to advance, at each step, in the direction of the gradient. Therefore, this method is somewhat complex and not the best method to use when attempting to optimize with a computer. An example of this is shown in Appendix 3.

3. Failure of Calculus - Direct Search

This type of search method differs from the previous method because direct search procedures work directly with the objective function and avoid the difficulties of calculating the gradients of functions (Nicholson 84). Advantages of this type of search method are that not only is it less complicated than the gradient search, but it also can readily cope with constraints (Nicholson 84). However, a disadvantage is that with an increasing number of variables (sources), the required time for computation may become large (Nicholson 84). This search method uses a base point and perturbations to slowly move toward the optimal point. An example of this procedure is included in Appendix 4.

4. Exhaustive Search

Since the previous methods are somewhat complex, a more computer-adaptable method, such as an exhaustive search, was considered. This method employs an examination of all possible combinations of k_i values, and then identifies the set(s) of k_i values that will optimize the objective function. Depending on the amount of accuracy desired in a solution, the number of calculations required to arrive at a solution can be easily determined. For example, if ± 1% accuracy is desired (i.e., k_i = 0.00, 0.01, ..., 1.00), then each source (assuming a full range of 0.00 - 1.00) would have 101 different possible k_i values. The following table shows the number of combinations that must be examined for an exhaustive analysis (assuming n-1 independent sources):

Number of Sources	**Possible Solutions**
2	101
3	10, 201
4	1, 030, 301
⋮	⋮
9	1.08×10^{16}
10	1.09×10^{18}

However, should ± 0.01% accuracy be desired, each source will have 10,001 possible values. The number of feasible solutions then increases considerably, as shown in the following table (assuming n-1 independent sources):

Number of Sources	Possible Solutions
2	10, 001
3	100, 020, 001
4	1.00×10^{12}
⋮	⋮
9	1.00×10^{32}
10	1.00×10^{36}

This method of searching has the disadvantage of being computationally intensive, even for a fast computer. However, there are several advantages of this method. First, it is simple to execute. A computer simply uses loops to search over the entire range for each variable, comparing the current solution to the currently optimal solution. Second, this method guarantees optimality within a certain limit of accuracy. Because every possible combination is considered, optimality can be assured. It is the tremendous speed and error-free calculation ability of the computer that this exhaustive search method seeks to exploit (Conley 5).

5. Iterative Search (Strategic Guessing)

Although the exhaustive search will work in all cases as well as guarantee optimality, it must be admitted that much of the computational effort amounts to sheer waste. Many of the calculations can quickly be discarded as non-optimal. However, is there a way for the computer to do this? Specifically, if an algorithm can quickly eliminate a number of answers, then very large problems can be solved with substantially fewer calculations. For example, as indicated earlier, an exhaustive search for ten variables ranging from 0-100% at 0.01% increments would require about 10^{36} calculations, disregarding the comparisons. However, what if an algorithm was able to make a few

"smart" decisions and reduce the number of calculations to 10^{10} ? This concept of looking only at strategic points and using those values to make decisions is the driving force behind design of experiments (DOE). However, DOE can be used to adequately understand the response surface, then assumes that optimization will easily follow. For this project, the response surface, M_p, is known and its optimization is the problem. Nevertheless, the experimentation methodology may be used to strategically locate the optimal point. Cornell explains that a mixture problem with n ingredients results in a factor space called a simplex. The geometric description of the factor space containing the n components consists of all points on or inside the boundaries (vertices, edges, faces, etc.) of a regular (n-1)-dimensional simplex (Cornell 6). Three sources (n = 3) result in a simplex factor space that is an equilateral triangle, and for n = 4 the simplex is a tetrahedron (Cornell 6). Therefore, the next step is to determine a way to selectively examine points in the simplex; one possible approach is to evenly divide the simplex, taking observations throughout. An ordered arrangement consisting of a uniformly spaced distribution of points on a simplex is known as a lattice (Cornell 22). A lattice design can be more accurate as the number of points in the simplex is increased. A lattice design is said to have degree m where the points in the simplex are spread such that the values of each component are

$$k_i = 0, \frac{1}{m}, \frac{2}{m}, \cdots, 1$$

if $0 \le k_i \le 1$. Thus, a lattice design may be categorized by its degree and by the number of components in the mixture. The number of selected points in a $\{n,m\}$ simplex-lattice design will be

$$\frac{(n+m-1)!}{m!(n-1)!}.$$

For example, a {3,2} design will have 6 points, while a {10,4} design will have 715 points. A table showing the total number of points for a $\{n,m\}$ lattice is shown in Appendix 5. Then, a further lattice could be used on a selected region of the first simplex, so that the optimal point is slowly brought to focus. An example with n = 2 is shown in Appendix 6.

6. Iterative Search (Monte Carlo Simulation)

This was the method selected for use with the blending problem. Instead of attempting to equally divide the simplex into regions as is done with an iterative search method, a Monte Carlo simulation method examines a random sample of points inside the simplex (Conley 20). Based on the optimal value in the sample, the simplex region can be halved and another random sample of points can be taken. The computer can select the optimal sample value at each iteration to be the centroid for the next iteration's simplex, halving the simplex region as well. This technique employs statistical probability and then relies on the massive speed of the computer to determine an optimal solution.

As with any optimization technique, both advantages and disadvantages exist. A primary advantage is that the method uses far fewer calculations than an exhaustive search. Conley stated (as of 1981) that a computer, using Monte Carlo techniques, found a true optimum solution to a problem with 10^{30} feasible solutions in about a minute on a medium-sized computer, where an exhaustive search technique on a problem of this scale would take the computer about 32 million trillion years (Conley 20). Other advantages include the ability to use an accurate (nonlinear) model with no concern for linearity, continuity, or differentiability assumptions. One disadvantage is that optimality cannot be assured because the method is based on randomness, and the method could conceivably miss an optimal point. However, the probability that a Monte Carlo solution investigating 1,000,000 combinations will come within 0.00001 of the optimal solution is $1 - 0.99999^{1,000,000} = 0.9999546$. Therefore, the probability is almost overwhelming that at least a near optimum solution will be found.

The basis for this method is that all feasible solutions map a distribution of objective function values (see Appendix 7). Most values tend toward the objective function mean, and the optimal solution (to the specific problem of this project, M_p) is at the left tail of the distribution. At each iteration of the simulation, a specified number of samples, say 4000, are selected. The probability that one random sample will *not* produce a value in the lower 0.01 tail of the M_p distribution is 0.99. However, the probability that *none* of the 4000 samples will produce a value in the lower 0.01 tail of

the distribution is 0.99^{4000}, which is about 3.5×10^{-18}. With each iteration, the feasible region is made smaller and smaller, so that another 4000 samples will most likely yield a solution in an even lower tail of the distribution, say 0.0001. After several iterations (≤ 10), the optimal solution is virtually guaranteed. Therefore the optimal solution will most likely be found after the computer evaluates a mere $4000 \times 10 = 40{,}000$ possible solutions, which is a drastic reduction in calculations compared to the exhaustive search method. The only situation that would disrupt the Monte Carlo approach would be a distribution (in an optimization problem) that had one isolated optimal solution that was separate from the remainder of the solutions (Conley 189, see Appendix 8).

The optimization procedure works as follows:

1. Assign the initial optimal solution by letting $k_i = 1/n$ for $i = 1, 2, \ldots, n$, where $\Sigma k_i = 1$ and $0 \leq k_i \leq b_i$. Call this solution k_i^*. Its corresponding objective function value, M_p, will be M_p^*.
2. Set iteration number, $j = 1$.
   A. Randomly generate 4000 samples, where for sample s, $s = 1, 2, \ldots, 4000$, the k_i are randomly selected on the interval $0 \leq k_i \leq b_i$ and where $\Sigma k_i = 1$. Compute the corresponding M_p value for each sample.
   B. For each sample $s = 1, 2, \ldots, 4000$, compare M_p with M_p^*. If $M_p \leq M_p^*$, then replace M_p^* with M_p and let $k_i^* = k_i$.
   C. Using k_i^*, $i = 1, 2, \ldots, n$, as the centroid for source i, determine the new reduced range of each k_i for the next iteration where
   $$\max (k_i^* - 2^{-j-1}, 0) \leq k_i \leq \min (k_i^* + 2^{-j-1}, b_i).$$
3. For iterations $j = 2, 3, \ldots, 10$, use k_i^* and M_p^* from the previous iteration.
   A. Randomly generate 4000 samples, where for sample s, $s = 1, 2, \ldots, 4000$, the k_i are randomly selected on the interval
   $$\max (k_i^* - 2^{-j-1}, 0) \leq k_i \leq \min (k_i^* + 2^{-j-1}, b_i)$$

obtained from the previous iteration, where $\Sigma k_i = 1$ and a corresponding M_p is computed.

B. For sample $s = 1, 2, \ldots, 4000$, compare M_p with M_p^*. If $M_p \leq M_p^*$, then replace M_p^* with M_p and let $k_i^* = k_i$.

C. Using k_i^*, $i = 1, 2, \ldots, n$, as the centroid for source i, determine the new reduced range of each k_i for the next iteration where

$$\max (k_i^* - 2^{-j-1}, 0) \leq k_i \leq \min (k_i^* + 2^{-j-1}, b_i)$$

where j indicates the iteration just completed.

D. Return to step 3 until 10 iterations have been completed.

4. The identified optimal solution is k_i^*, with an objective function value of M_p^*.

For example, assume that the problem consists of three sources, that the optimal solution is 0.1, 0.3, and 0.6 for each source respectively, resulting in an objective function value of 50, and that each source has bounds placed at 0.0 and 0.8:

Step 1. The initial solution (k_i^*) would be set at 0.3333 for each source, with an objective function value of, say, $M_p^* = 62$.

Step 2 (**A**) The algorithm would generate the first sample of proportion values, say $k_1 = 0.2289$, $k_2 = 0.5372$, and $k_3 = 0.2339$, and (**B**) evaluate the objective function to be $M_p = 63.87$. Therefore, the second random sample would be generated and again compared to the initial optimal solution. Assume that after 4000 samples, the best solution thus far is $k_1^* = 0.1019$, $k_2^* = 0.2901$, and $k_3^* = 0.6080$, with an objective function value of $M_p^* = 52.17$. (**C**) This solution would form the basis for determining the new range of possible values. For iteration 1, the algorithm attempts to add/subtract $2^{-2} = 0.25$ from each k_i^* value to obtain the new reduced range. The first source would now have a range of (max (0.1019 - 0.25, 0.00)) = 0.00 and (min (0.1019 + 0.25, 0.8000)) = 0.3519, the second a range of 0.0401 - 0.5401, and the third a range between 0.3580 - 0.8000. Thus, the second source has a full range of 0.5000 while the other two were limited by their bounds, resulting in smaller ranges.

Step 3 **(A/B)** Iteration 2 would then begin and another 4000 samples would be generated within this smaller range of k_i values. **(C)** Assume this time that after the 4000 additional samples, the best proportions found thus far are 0.1009, 0.2987, and 0.6004, with an objective function value of $M_p^* = 50.19$. This solution would form the basis for determining the new range of possible values. For each source, the algorithm now attempts to add/subtract $2^{-3} = 0.125$ from each k_i^* value to obtain the new reduced range. The first source would now have a range of 0.0000 - 0.2269, the second a range of 0.1737 - 0.4237, and the third a range of 0.4754 - 0.7254. This time, the second and third sources have a maximum range of 0.25 while the first has a smaller range. **(D)** The second iteration would be complete and eight more iterations would follow, each taking 4000 samples from an ever-decreasing range. At iteration 10, the range for each of the sources would be $2^{-9} = 1.91 \times 10^{-3}$. This means that 4000 random samples would be selected from this range on each of the k_i values, which would guarantee optimality with acceptable accuracy.

Step 4 The optimal solution of $k_i^* = (0.1000, 0.3000, 0.6000)$ for $i = 1, 2, 3$, with an objective function value of $M_p^* = 50.00000$ is reported as the final answer.

At each iteration, the feasible region is at least halved and most likely reduced much more. If j is the iteration number, where $j = 1, 2, \ldots, 10$, and $j = 1$ (first iteration), with $k_1^* = 0.7$ and the bounds $0.00 \le k_1 \le 0.90$, then for the next iteration (i.e., $j = 2$),

$$\max (0.7 - 2^{-1-1}, 0.00) \le k_1 \le \min (0.70 + 2^{-1-1}, 0.90)$$

$$\max (0.7 - 0.25, 0.00) \le k_1 \le \min (0.70 + 0.25, 0.90)$$

$$\max (0.45, 0.00) \le k_1 \le \min (0.95, 0.90)$$

$$0.45 \le k_1 \le 0.90.$$

Essentially, the procedure uses $(n\text{-}1)$-dimensional "rectangles" that search through the n dimensional space (always staying inside the constraints) until the rectangles focus on the optimal answer (Conley 229). According to the laws of probability, the algorithm will quickly find the optimal solution.

The Monte Carlo method disregards classical optimization methods in favor of relying on the pure speed of the computer combined with the laws of probability. For any optimization problem of one hundred variables or less, Conley (245) recommends using this method to optimize a function.

MONTE CARLO SEARCH COMPUTER PROGRAM

The program has two options: 1) optimize a blend, and 2) user "What-if" analysis. The first option will allow the user to input the following values:

1. Number of sources, n (selected from the range 2-10)
2. For each source:
    A. Source mean, μ_i
    B. Source variation : variance, σ_i^2 or standard deviation, σ_i
    C. Source upper bound, b_i (1.00 = default value); (b_n must remain at 1.00)
3. "Safe limit," S (used for comparison)
4. Probability p (proportion of actual blend output that will exceed "maximum" value, M_p*. The program will then determine the appropriate number of standard deviations, Z_p, to add to the expected mean of the blend to correspond with this probability. Note that possible p values are: 0.25, 0.10, 0.05, 0.025, 0.01, 0.005, 0.0025, 0.001, 0.0001).

The program will output:

1. The expected output of the blend, μ_B
2. The standard deviation of the blend, σ_B
3. The optimal minimum "max" value, M_p*
4. A comparison of optimal and "Safe limit" (SAFE/UNSAFE)
5. The corresponding k_i* for each source

The second option of the program will be for user "What-if" analysis. The user inputs will be:

1. Number of sources, n (selected from the range 2-10)
2. For each source:

A. Source mean, μ_i

B. Source variation: variance, σ_i^2 or standard deviation, σ_i

C. Proportion of the source, k_i, in the blend (in keeping with $\Sigma k_i = 1$)

3. "Safe limit" value, *S*

The program will then give:

1. The expected output value of the given blend, μ_B
2. The standard deviation of the blend, σ_B
3. Proportion of the blend output, *B*, that will exceed the "Safe limit."

VISUAL BASIC 4.0

Microsoft's *Visual Basic 4.0 for Windows* was the programming language selected to code the computer program. "Visual Basic is an object-oriented/event-driven programming language that is easy enough for a nonprogrammer to use, yet sophisticated enough to be used by professional programmers" (Zak 5). Although Visual Basic is a relatively new language, several advantages that the language offers made this selection practical. Among the advantages are:

- Visual Basic programs are run in a Windows environment, making the user interface simple, neat, and user-friendly
- Visual Basic programs can be stored in an executable file (*.exe) so that any user with a *Windows 95* operating system may run the program
- Visual Basic allows the programmer to spend more time in coding the program details while user interface is virtually self-coded by the program itself.

A Visual Basic program is structured around forms and objects. Forms are the different windows that will allow user interaction, and the objects are the various text boxes, drop menus, and click commands that appear on each form. Therefore the first step in Visual Basic programming is to develop the user interface in the framework of forms and objects. After this major step, instructions concerning each object on the form are given as code.

This program has four basic forms. The first form is the "Control Panel." From this form, the user may move to one of the major program functions or exit the program. The second major form is the "Optimization" form. This form contains the overall inputs and all outputs for the optimization problem. The third form is the "What-if" analysis form which contains overall inputs and all outputs for the "What-if" analysis. The last form actually contains a sub-form for each source. These sub-forms allow the user to input data specific to each blending source.

In order to allow the reader and perhaps others who wish to expand on this project to more fully understand the program as a whole and the optimization procedure, pseudocode is given as Appendix 9. Furthermore, the actual code for the optimization of the blending problem with two sources is given as Appendix 10, while Appendix 11 contains the code used in the "What-if" analysis for two sources. The reader will note that all *Visual Basic 4.0* "comments" are preceded by an apostrophe (').

PROGRAM VALIDATION

In order to verify whether Blend was properly optimizing proportions, nine sample problems (one for each $n = 2,...,10$) were optimized. Before optimizing with Blend, the problems were solved with LINGO, a nonlinear optimization software package. Each problem was run five times while comparing only the M_p^* values, since it is true that the actual proportions can vary substantially while maintaining extremely similar M_p^* values. From all 45 runs, the average deviation from the optimal answer was 0.66%. Of interest was the fact that most of the error came from one problem that Blend seemed to have more difficulty in optimizing, while no unique features of the problem could be established. Overall, Blend demonstrated a strong optimization potential; experimental results for problems where $n = 3, 7$, and 10 are shown in Appendix 12.

REFERENCED LITERATURE

Conley, William. Optimization: A Simplified Approach. Petrocelli Books, Inc., NY, 1981.

Cornell, John A. Experiments with Mixtures. 2nd ed. John Wiley & Sons, Inc., NY, 1990.

Hillier, Frederick S., and Gerald J. Lieberman. Introduction to Operations Research. 5th ed. McGraw-Hill, USA, 1990.

Nicholson, T.A.J. Optimization in Industry. Volume 1: Optimization Techniques. Aldine·Atherton, Inc., Chicago, 1971.

Zak, Diane. Programming with Microsoft Visual Basic 4.0 for Windows. CTI, Cambridge, MA, 1997.

UNREFERENCED LITERATURE

Chvátal, Vasek. Linear Programming. W.H. Freeman and Company, NY, 1983.

Foulds, L.R. Optimization Techniques: An Introduction with 72 Illustrations. Springer-Verlag, NY, 1981.

Gabasov, Rafail F., and Faina M. Kirillova. Methods of Optimization. Optimization Software, Inc., NY, 1988.

Hines, William W. and Douglas C. Montgomery. Probability and Statistics in Engineering and Management Science. 3rd Ed. John Wiley & Sons, NY, 1990.

Nicholson, T.A.J. Optimization in Industry. Volume 2: Industrial Applications. Aldine·Atherton, Inc., Chicago, 1971.

Penfold, John W. Microsoft Visual Basic: The Programmer's Companion. Sigma Press, England, 1993.

Prata, Stephen. Certified Course in Visual Basic 4. Waite Group Press, CA, 1995.

Wang, Wallace. Visual Basic 4 for Windows for Dummies. IDG Books Worldwide, USA, 1996.

APPENDIX 1

General Formula Using (n-1) Partial Derivatives

General Formula Using (n-1) Partial Derivatives

For this derivation, assume:

Source	Proportion	Mean	Standard Deviation
1	k_1	$s_1\mu_1$	$r_1\sigma_1$
2	k_2	$s_2\mu_1$	$r_2\sigma_1$
⋮	⋮	⋮	⋮
n-1	k_{n-1}	$s_{n-1}\mu_1$	$r_{n-1}\sigma_1$
n	$1-\sum_{i=1}^{n-1} k_i$	$s_n\mu_1$	$r_n\sigma_1$

and where $s_1 = r_1 = 1$. Let the subscript B denote the blend.

Blend Mean:

$$\mu_B = k_1 s_1 \mu_1 + k_2 s_2 \mu_1 + \cdots + k_{n-1} s_{n-1} \mu_1 + \left(1 - \sum_{i=1}^{n-1} k_i\right) s_n \mu_1$$

$$\mu_B = \mu_1 \left[k_1 (s_1 - s_n) + k_2 (s_2 - s_n) + \cdots + k_{n-1} (s_{n-1} - s_n) + s_n\right]$$

$$\mu_B = \mu_1 \left[\sum_{i=1}^{n-1} k_i (s_i - s_n) + s_n\right]$$

Blend Standard Deviation:

$$\sigma_B^2 = k_1^2 r_1^2 \sigma_1^2 + k_2^2 r_2^2 \sigma_1^2 + \cdots + k_{n-1}^2 r_{n-1}^2 \sigma_1^2 + \left(1 - \sum_{i=1}^{n-1} k_i\right)^2 r_n^2 \sigma_1^2$$

$$\sigma_B = \sigma_1 \left[\sum_{i=1}^{n-1} k_i^2 r_i^2 + r_n^2 \left(1 - \sum_{i=1}^{n-1} k_i\right)^2\right]^{1/2}$$

Now, define $M_p = \mu_B + Z\sigma_B$

$$M_p = \mu_1 \left[\sum_{i=1}^{n-1} k_i (s_i - s_n) + s_n\right] + Z\sigma_1 \left[\sum_{i=1}^{n-1} k_i^2 r_i^2 + r_n^2 \left(1 - \sum_{i=1}^{n-1} k_i\right)^2\right]^{1/2}$$

Taking the partial derivative results in (n-1) equations:

$$\frac{\partial M_p}{\partial k_j} = \mu_1\left(s_j - s_n\right) + \frac{Z}{2}\sigma_1\left[\sum_{i=1}^{n-1} k_i^2 r_i^2 + r_n^2\left(1-\sum_{i=1}^{n-1} k_i\right)^2\right]^{-\frac{1}{2}} \cdot \left(2k_j r_j^2 + 2k_j r_n^2 - 2r_n^2 + 2r_n^2\left(\sum_{i=1}^{n-1} k_i - k_j\right)\right)$$

$$\forall j = 1,2,\ldots,n-1$$

$$\frac{\partial M_p}{\partial k_j} = \mu_1\left(s_j - s_n\right) + Z\sigma_1\left(k_j r_j^2 + r_n^2\left(k_j - 1 + \sum_{i=1}^{n-1} k_i - k_j\right)\right)\left[\sum_{i=1}^{n-1} k_i^2 r_i^2 + r_n^2\left(1-\sum_{i=1}^{n-1} k_i\right)^2\right]^{-\frac{1}{2}}$$

$$\forall j = 1,2,\ldots,n-1$$

$$\frac{\partial M_p}{\partial k_j} = \mu_1\left(s_j - s_n\right) + Z\sigma_1\left(k_j r_j^2 + r_n^2\left(\sum_{i=1}^{n-1} k_i - 1\right)\right)\left[\sum_{i=1}^{n-1} k_i^2 r_i^2 + r_n^2\left(1-\sum_{i=1}^{n-1} k_i\right)^2\right]^{-\frac{1}{2}}$$

$$\forall j = 1,2,\ldots,n-1$$

$$\frac{\partial M_p}{\partial k_j} = \mu_1\left(s_j - s_n\right) + Z\sigma_1\left(k_j r_j^2 - r_n^2\left(1-\sum_{i=1}^{n-1} k_i\right)\right)\left[\sum_{i=1}^{n-1} k_i^2 r_i^2 + r_n^2\left(1-2\sum_{i=1}^{n-1} k_i + \sum_{i=1}^{n-1}\sum_{m=1}^{n-1} k_i k_m\right)\right]^{-\frac{1}{2}}$$

$$\forall j = 1,2,\ldots,n-1$$

APPENDIX 2

Example of an Analytical Solution when $n = 2$ Sources

Example of an Analytical Solution when $n = 2$ Sources

Given :

$\mu_1 = 50$ $\sigma_1 = 2$ $\mu_2 = 40$ $\sigma_2 = 6$

assume

$Z_\alpha = 0.00135$, so $Z = 3.00$

then

$s_1 = 1$ $r_1 = 1$ $s_2 = .8$ $r_2 = 3$

---

solution will require $(n - 1) = 1$ derivative :

$$\frac{\partial M_p}{\partial k_j} = \mu_1\left(s_j - s_n\right) + Z\sigma_1\left(k_j r_j^2 - r_n^2\left(1 - \sum_{i=1}^{n-1} k_i\right)\right)\left[\sum_{i=1}^{n-1} k_i^2 r_i^2 + r_n^2\left(1 - 2\sum_{i=1}^{n-1} k_i + \sum_{i=1}^{n-1}\sum_{m=1}^{n-1} k_i k_m\right)\right]^{-1/2}$$

$$\frac{\partial M_p}{\partial k_1} = 50\left(1 - 0.8\right) + 3 \cdot 2\left(k_1 \cdot 1^2 - 3^2\left(1 - k_1\right)\right)\left[k_1 \cdot 1^2 + 3^2\left(1 - 2k_1 + k_1^2\right)\right]^{-1/2}$$

set derivative equal to zero and solve for k_1 :

$$\frac{\partial M_p}{\partial k_1} = 10 + \left(60k_1 - 54\right)\left[10k_1^2 - 18k_1 + 9\right]^{-1/2} = 0$$

$$-10\sqrt{10k_1^2 - 18k_1 + 9} = 60k_1 - 54$$

$$100\ (\ 10k_1^2 - 18k_1 + 9) = 3600k_1^2 - 6480k + 2916$$

$$26k_1^2 - 46.8k_1 + 20.16 = 0$$

using the quadratic formula :

$$k_1 = \frac{46.8 \pm \sqrt{46.8^2 - 4 \cdot 26 \cdot 20.16}}{2 \cdot 52}$$

$$k_1 = 1.086 \quad \underline{\text{or}} \quad 0.7139$$

then

$$k_1 = 0.7139 \quad \text{and} \quad k_2 = 0.2861$$

APPENDIX 3

Example of Gradient Search Procedure ($n = 2$)

Example of Gradient Search Procedure ($n = 2$)
(adapted from Hillier/Lieberman)

Given :

$\mu_1 = 50$ $\sigma_1 = 2$ $\mu_2 = 40$ $\sigma_2 = 6$

assume

$\varepsilon = .005$ $Z_\alpha = 0.00135$, so $Z = 3.00$

then

$s_1 = 1$ $r_1 = 1$ $s_2 = .8$ $r_2 = 3$

$$M_p = 10k_1 + 40 + 6\sqrt{10k_1^2 - 18k_1 + 9}$$

$$\frac{dM_p}{dk_1} = 10 + (60k_1 - 54)(10k_1^2 - 18k_1 + 9)^{-1/2}$$

---

begin at $k_1 = 0.5$ $\quad \dfrac{dM_P}{dk_1} = -0.7331$

$$\nabla M_p = (-0.7331)$$

$$k_1 = 0.5 - t\,(-0.7331) = 0.5 + 0.7331t$$

$$M_p = 10(0.5 + 0.7331t) + 40 + 6\sqrt{10(0.5 + 0.7331t)^2 - 18(0.5 + 0.7331t) + 9}$$

$$M_p = 45 + 7.331t + 6\sqrt{2.5 + 7.331t + 5.3744t^2 - 9 - 13.1958t + 9}$$

$$M_p = 45 + 7.331t + 6\sqrt{5.3744t^2 - 5.8648t + 2.5}$$

one-dimensional search to find min ($t \geq 0$) t^* of M_p

$$\frac{dM_p}{dt} = 7.331 + \frac{6 \cdot 0.5(10.7488t - 5.8648)}{\sqrt{5.3744t^2 - 5.8648t + 2.5}} = 0$$

$$\frac{32.2464t - 17.5944}{\sqrt{5.3744t^2 - 5.8648t + 2.5}} = -7.331$$

$$(32.2464t - 17.5944)^2 = (-7.331)^2(5.3744t^2 - 5.8648t + 2.5)$$

$$1039.8303t^2 - 1134.7121t + 309.5629 = 288.8396t^2 - 315.1955t + 134.359$$

$$750.9907t^2 - 819.5166t + 175.2039 = 0$$

$$t = 0.7994 \quad \underline{\text{or}} \quad t = 0.2918$$

$k_1 = 0.5 + 0.7331(0.7994)$ or $k_1 = 0.5 + 0.7331(0.2918)$

$= 1.086$ $= 0.7139$

* not feasible * feasible

$$\nabla M_p(0.7139) = \frac{dM_p}{dk_1} = 10 - \frac{11.166}{\sqrt{1.2463}} = -0.002$$

which is sufficiently close since $\varepsilon = .005$.

Therefore, $k_1 = 0.7139$ and $k_2 = 0.2861$.

APPENDIX 4

Example of Direct Search Procedure

Example of Direct Search Procedure
(adapted from Nicholson)

Given :

$\mu_1 = 50$ $\sigma_1 = 2$ $\mu_2 = 40$ $\sigma_2 = 6$

assume

$\varepsilon = .01$ $Z = 3.00$

then

$s_1 = 1$ $r_1 = 1$ $s_2 = .8$ $r_2 = 3$

$$M_p = 10k_1 + 40 + 6\sqrt{10k_1^2 - 18k_1 + 9}$$

---

begin at $k_1 = 0.5$ perturbation step size, $\delta = 0.01$

*Base point, $B^{(0)} = 0.5$

$M_p\ (B^{(0)}) = 54.4868$

$M_p\ (B^{(0)} + \delta) = 54.4357$ accept and let $T_o^{(1)} = 0.51$

* check "double" step $M_p\ (B^{(0)} + 2\delta) = 54.3861$ accept $T_1^{(1)} = 0.52$

* check "double" step $M_p\ (B^{(0)} + 2(2\delta)) = 54.2913$ accept $T_2^{(1)} = 0.54$

* check "double" step $M_p\ (B^{(0)} + 2(4\delta)) = 54.1225$ accept $T_3^{(1)} = 0.58$

* check "double" step $M_p\ (B^{(0)} + 2(8\delta)) = 53.8894$ accept $T_4^{(1)} = 0.66$

* check "double" step $M_p\ (B^{(0)} + 2(16\delta)) = 54.0910$ reject

so $T_4^{(1)} = 0.66 = B^{(1)}$

* begin again

$M_p\ (B^{(1)} + \delta) = 53.8724$ accept $T_0^{(2)} = 0.67$

* check "double" step $M_p\ (B^{(1)} + 2\delta) = 53.8586$ accept $T_1^{(2)} = 0.68$

* check "double" step $M_p\ (B^{(1)} + 2(2\delta)) = 53.8411$ accept $T_2^{(2)} = 0.70$

* check "double" step $M_p\ (B^{(1)} + 2(4\delta)) = 53.851$ reject

so $T_2^{(2)} = 0.70 = B^{(2)}$

* begin again

$M_p\,(B^{(2)} + \delta) = 53.8377$ accept $T_0^{(3)} = 0.71$

* check "double" step $M_p\,(B^{(2)} + 2\delta) = 53.8381$ reject

so $T_0^{(3)} = 0.71 = B^{(3)}$

* begin again

$M_p\,(B^{(3)} + \delta) = 53.8381$ reject

* since no step, try step back

$M_p\,(B^{(3)} - \delta) = 53.8411$ reject

* since step in either direction is worse, take current base as optimal

$B^{(3)} = 0.71$ so $k_1 = 0.71$ $k_2 = 0.29$

APPENDIX 5

Total Points of a Lattice Design

Total Points of a Lattice Design

$n\rightarrow$ $m\downarrow$	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
1	2	3	4	5	6	7	8	9	10
2	3	6	10	15	21	28	36	45	55
3	4	10	20	35	56	84	120	165	220
4	5	15	35	70	126	210	330	495	715
5	6	21	56	126	252	462	792	1287	2002

APPENDIX 6

Example of a Lattice Design Search Procedure ($n = 2$)

Example of a Lattice Design Search Procedure ($n = 2$)

Given :

$\mu_1 = 50$ $\sigma_1 = 2$ $\mu_2 = 40$ $\sigma_2 = 6$

assume

$\varepsilon = .005$ $Z = 3.00$ lattice order = 5

then

$s_1 = 1$ $r_1 = 1$ $s_2 = .8$ $r_2 = 3$

$$M_p = 10k_1 + 40 + 6\sqrt{10k_1^2 - 18k_1 + 9}$$

---

* search for optimal k_1 beginning with endpoints at (0,1)

$M_p(0)$ =	58
$M_p(0.25)$ =	56.08
$M_p(0.50)$ =	54.49
$M_p(0.75)$ =	53.86 *
$M_p(1)$ =	56

* re-center search and continue ...

$M_p(0.50)$ =	54.49
$M_p(0.625)$ =	53.97
$M_p(0.75)$ =	53.86 *
$M_p(0.875)$ =	54.46
$M_p(1)$ =	56

* re-center search and continue ...

$M_p(0.625)$ =	53.97
$M_p(0.6875)$ =	53.850 *
$M_p(0.75)$ =	53.86
$M_p(0.8125)$ =	54.054
$M_p(0.875)$ =	54.46

* re-center search and continue ...

$M_p(0.625)$ =	53.97
$M_p(0.65625)$ =	53.8966
$M_p(0.6875)$ =	53.850
$M_p(0.71875)$ =	53.8378 *
$M_p(0.75)$ =	53.86

* re-center search and continue ...

$M_p(0.6875) =$ 53.850
$M_p(0.703125) =$ 53.83959
$M_p(0.71875) =$ 53.8378 *
$M_p(0.734375) =$ 53.8457
$M_p(0.7500) =$ 53.86

* re-center search and continue ...

$M_p(0.703125) =$ 53.83959
$M_p(0.7109375) =$ 53.83753 *
$M_p(0.71875) =$ 53.8378
$M_p(0.7265625) =$ 53.8405
$M_p(0.734375) =$ 53.8457

* re-center search and continue ...

$M_p(0.703125) =$ 53.83959
$M_p(0.70703125) =$ 53.83827
$M_p(0.7109375) =$ 53.83753
$M_p(0.71484375) =$ 53.8373702 *
$M_p(0.71875) =$ 53.8378

* re-center search and continue ...

$M_p(0.7109375) =$ 53.83753
$M_p(0.712890625) =$ 53.8373763
$M_p(0.71484375) =$ 53.8373702 *
$M_p(0.716796875) =$ 53.8375129
$M_p(0.71875) =$ 53.8378

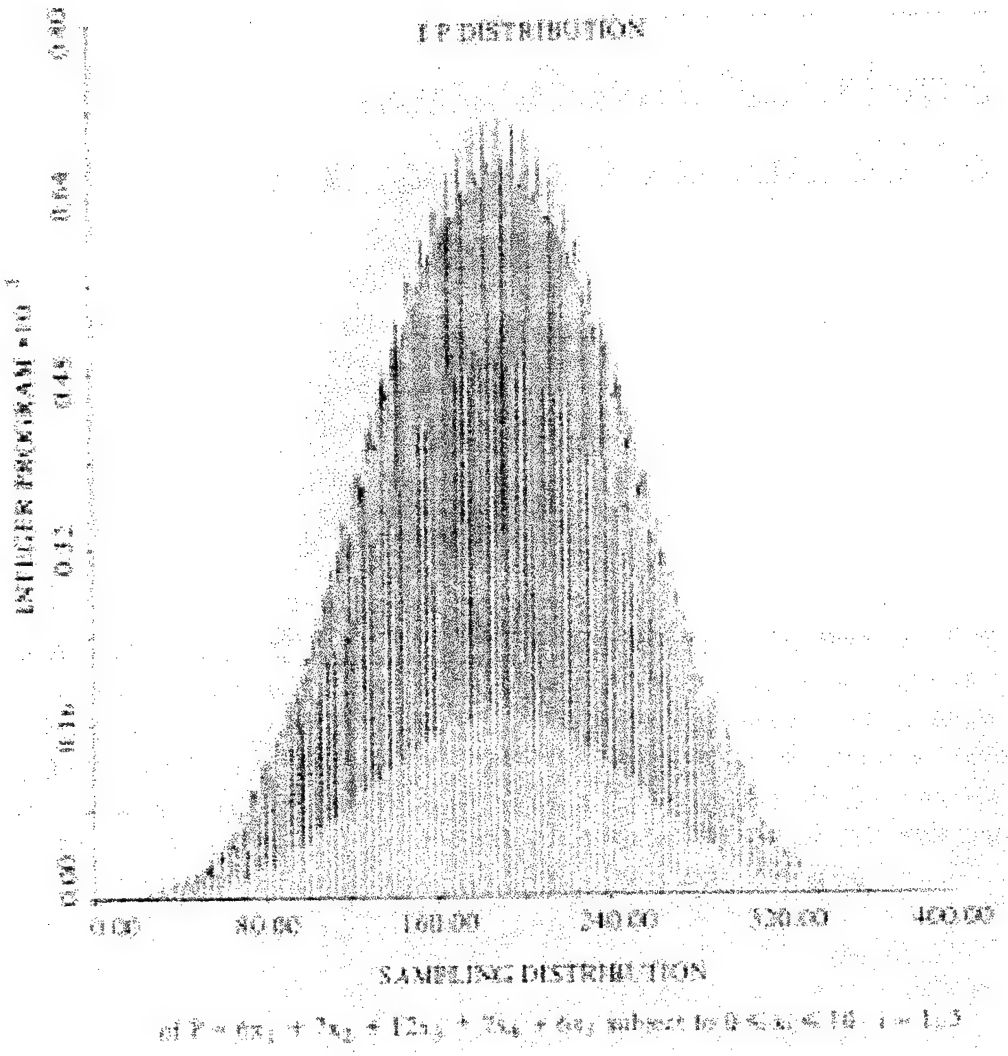
accuracy currently at 0.001953125, which is sufficient since $\varepsilon = .005$.

$k_1 = 0.71484375$ and $k_2 = 0.28515625$

APPENDIX 7

Distribution of Objective Function Values

Distribution of Objective Function Values
(taken from Conley)

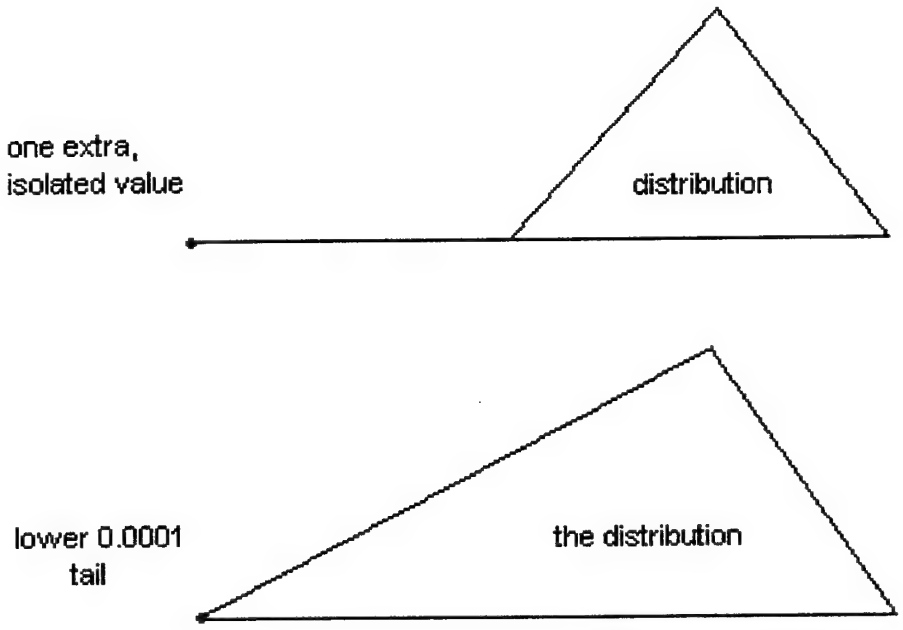


Appendix 7

APPENDIX 8

Disruption of a Monte Carlo Approach

Disruption of a Monte Carlo Approach



APPENDIX 9

Pseudocode

PSEUDOCODE

This appendix gives the general code for each object of each form (i.e. how the program will respond to any action that the user might take). Note that "dimming" involves leaving a label or text box in sight while not allowing the user to access it. Also, the mouse "focus" is defined as the current cursor position on a form.

1. Control Panel

When the user clicks on "Optimize"

Remove the "What-if" form from view, should it currently be active

Show the "Optimize" form while also showing the Source Data form minimized

When the user clicks on "What-if"

Remove the "Optimize" form from view, should it currently be active

Show the "What-if" form while also showing the Source Data form minimized

When the user clicks on "Exit"

Exit the program

2. Optimize

When the form loads

Initialize the "Already" boolean variable to false

Load the choices into the "Number of Sources" list box

Initialize the box set to Number Sources = 2

Load the choices into the "Probability" list box

Initialize the box set to Probability = 0.01

Call the "Dim Blend Proportions" procedure

Call the "Un-dim Bounds" procedure

When the user changes the Number of Sources using the list box

Hide/Disable Source Data forms and Optimal Proportions (sources 3-10)

Un-dim all upper bounds for sources 3-9

Show appropriate Source Data forms and Optimal Proportions (sources 3-10)

Show warning message about forms 8-9-10 if necessary (Already is false) and set Already to true

When the user clicks on the "Cancel" button

Close Source Data/Optimize forms

Show the Control Panel form

When the user clicks on the "Optimize" button

Declare local variables

Initialize variables

Call appropriate optimization procedure (depending on number of sources)

Display output values

When the user changes the "Safe limit"

Error-check to ensure that a number was input

Show warning message if user did not input a numerical value

Return mouse focus to the "Safe limit" text box

3. "What-if" Analysis

When the form loads

Initialize the "Already" boolean variable to false

Load the choices into the "Number of Sources" list box

Initialize the box set to Number Sources = 2

Call the "Un-dim Blend Proportions" procedure

Call the "Dim Bounds" procedure

When the user changes the Number of Sources using the list box

Hide/Disable Source Data forms (for sources 3-10)

Show appropriate Source Data forms (for sources 3-10)

Show warning message about forms 8-9-10 if necessary (Already is false) and set Already to true

When the user clicks on the "Cancel" button

Close Source Data / What-if Analysis forms

Show the Control Panel form

When the user clicks on the "Compute" button

Declare local variables

Initialize variables

Call appropriate compute procedure (depending on number of sources)

Display output values

When the user changes/finishes changing the Maximum Safety Value text box

Error-check the input to ensure that it is a numerical value

Return the mouse focus back to the Max Safety Value text box

4. Source Data (10 forms, all identical)

When the form loses the mouse focus

Declare local variables

Error-check all source data (that all values are numerical)

Display an appropriate message if error exists

When the user changes/finishes changing any of the following: blend proportion, mean output, output variation, upper bound, lower bound

Declare local variables

Error-check the appropriate data (to ensure that it is a numerical value)

Check to ensure that numbers are feasible

Display error message if necessary

Return mouse focus to text box found in violation

5. Procedures

Dim Blend Proportions Procedure

Dim the blend proportion label and text box on each Source Data form

Dim Bounds Procedure

Dim the bound frame, labels, and text boxes on each Source Data form

Un-dim Blend Proportions Procedure

Un-dim the blend proportion label and text box on each Source Data form

Un-dim bounds Procedure

Un-dim the bound frame, labels, and text boxes on each Source Data form

Max Procedure

Set the maximum value to the first element in the array

Compare each element in the array, setting it to maximum value if it is larger

Min Procedure

Set the minimum value to the first element in the array

Compare each element in the array, setting it to minimum value if it is smaller

Z-Alpha Procedure

Select the appropriate case of the user-defined probability

Assign the Z value corresponding to that probability

Standard Normal Inverse Procedure

Select the appropriate range of standardized Z values

Assign the probability corresponding to that range of Z values

Compute "What-if" Analysis Procedure (9 procedures, each for a given number of sources)

Declare local variables

Initialize variables (including proportions (k_i), means, and variances for each source)

Determine if all proportions sum to 1, and if all data is non-negative

If error exists, display appropriate message and exit subroutine

Use a loop to calculate the blend's mean, variance, and standard deviation

If the standard deviation is zero, then

  Set the probability of exceeding maximum safe value to either 0 or 1

Otherwise,

  Calculate Z

  Call the Standard Normal Inverse procedure to find the probability

Optimize Procedure (9 procedures, each for a given number of sources)

Declare local variables

Initialize absolute bounds

Initialize source means/variances

Initialize Z, current conditions, and check initial centroid feasibility

Check upper bound feasibility (that the sum of upper bounds ≥ 1)

Set initial feasible solution

If feasible, then initial solution = centriod point (all sources set to 1/NumSources)

If not, then set all sources at upper bound until sum = 1

Compute mean, standard deviation, and objective function value of initial solution

For each iteration:

  For each sample:

    Set blend proportions (except last source) randomly within current bounds

    Set final blend proportion as (1-sum of others)

    Calculate objective function value, M_p

    Compare M_p to M_p*, replacing the optimal if M_p is less

    Repeat for all samples

    After all samples taken (i.e. end of iteration) reduce range of each blending source appropriately

  Repeat for all iterations

Calculate final values to send as output (blend mean, standard deviation, objective function value)

APPENDIX 10

Code for Optimization of 2 Sources

```
Public Sub OptTwo(OptMean, OptMax, OptStandDev, OptProb, NumSources, OptProport(
)
            As Single)

   ' *** THIS PROCEDURE IS THE CODE FOR OPTIMIZING 2 SOURCES ***


   ' DECLARE LOCAL VARIABLES
      Dim i As Integer                    ' index for sample size
      Dim j As Integer                    ' index for iterations
      Dim k As Integer                    ' index for each source
      Dim Z As Single                     ' Z alpha value for computing max value

      Dim AbsLow(1 To 2) As Single        ' absolute lower bounds for each source
      Dim AbsUp(1 To 2) As Single         ' absolute upper bounds for each source
      Dim Mu(1 To 2) As Single            ' mean output for each source
      Dim Var(1 To 2) As Single           ' variance for each source
      Dim OptimalMp As Single             ' current optimal max value
      Dim UBSum As Single                 ' error-checks upper bounds

      Dim BlendMean As Single             ' current blend's mean
      Dim BlendVar As Single              ' current blend's variance
      Dim CurrLow(1 To 2) As Single       ' current lower bound for each source
      Dim CurrRange(1 To 2) As Single     ' current feasible range for each source
      Dim CurrUp(1 To 2) As Single        ' current upper bound for each source
      Dim CurrSoln(1 To 2) As Single      ' current solution's proportions
      Dim CurrSolnMp As Single            ' current solution's max value
      Dim Sum As Single                   ' determines final source's proportion
      Dim Feasible As Boolean             ' determines initial centroid feasibility

   ' INITIALIZE VARIABLES
   ' initialize absolute bounds
      For k = 1 To NumSources
         AbsLow(k) = 0
      Next k
      AbsUp(1) = frmSourceOne!txtSourceOneUB.Text
      AbsUp(2) = frmSourceTwo!txtSourceTwoUB.Text
   ' initialize source means/variances
      Mu(1) = frmSourceOne!txtSourceOneMean.Text
      Mu(2) = frmSourceTwo!txtSourceTwoMean.Text
      If (frmSourceOne!optSourceOneSD.Value = True) Then
         Var(1) = (frmSourceOne!txtSourceOneVar.Text) ^ 2
      Else
         Var(1) = (frmSourceOne!txtSourceOneVar.Text)
      End If
      If (frmSourceTwo!optSourceTwoSD.Value = True) Then
         Var(2) = (frmSourceTwo!txtSourceTwoVar.Text) ^ 2
      Else
         Var(2) = (frmSourceTwo!txtSourceTwoVar.Text)
      End If
   ' initialize Z, current conditions; check centroid feasbility
      Feasible = True
      Z = ZAlpha(OptProb)
      UBSum = 0
      For k = 1 To NumSources
         CurrLow(k) = AbsLow(k)
         CurrUp(k) = AbsUp(k)
         UBSum = UBSum + AbsUp(k)
         If (AbsUp(k) < (1 / NumSources)) Then
            Feasible = False
         End If
      Next k
   ' check upper bound feasibility
      If (UBSum < 1) Then
         MsgBox "Upper bounds are infeasible; must sum to at least 1.", 48, "Upp
er Bounds"
         Exit Sub
```

```
Unoptimize = 2
    End If
  ' set initial feasible solution; compute corresponding objective function val
ue
    Sum = 0
    BlendMean = 0
    BlendVar = 0
    For k = 1 To NumSources
       If (Feasible = True) Then
          OptProport(k) = 1 / NumSources
       Else
          OptProport(k) = AbsUp(k)
          Sum = Sum + OptProport(k)
          If (Sum > 1) Then
             Sum = Sum - OptProport(k)
             OptProport(k) = 1 - Sum
             Sum = Sum + OptProport(k)
          End If
       End If
       BlendMean = BlendMean + (Mu(k) * OptProport(k))
       BlendVar = BlendVar + ((OptProport(k) ^ 2) * Var(k))
    Next k
    OptimalMp = BlendMean + Z * Sqr(BlendVar)

  ' CONDUCT OPTIMIZATION
    For j = 1 To NumIterations
       For i = 1 To SampleSize
  ' set blend proportions randomly
          Sum = 0
          For k = 1 To (NumSources - 1)
             CurrUp(k) = Min(CurrUp(k), 1 - Sum)
             CurrRange(k) = CurrUp(k) - CurrLow(k)
             CurrSoln(k) = CurrLow(k) + (CurrRange(k) * Rnd)
             Sum = Sum + CurrSoln(k)
          Next k
  ' set final source proportion as 1 - (sum of others)
          CurrSoln(NumSources) = 1 - Sum
  ' calculate objective function value (Mp)
          BlendMean = 0
          BlendVar = 0
          For k = 1 To NumSources
             BlendMean = BlendMean + (Mu(k) * CurrSoln(k))
             BlendVar = BlendVar + ((CurrSoln(k) ^ 2) * Var(k))
          Next k
          CurrSolnMp = BlendMean + Z * Sqr(BlendVar)
  ' compare to current optimal
          If CurrSolnMp <= OptimalMp Then
             OptimalMp = CurrSolnMp
             For k = 1 To NumSources
                OptProport(k) = CurrSoln(k)
             Next k
          End If
       Next i
  ' after all samples taken (i.e. end of iteration) reduce range of each
  ' blending source appropriately
       For k = 1 To NumSources
          CurrLow(k) = Max(OptProport(k) - 2 ^ (-j - 1), AbsLow(k))
          CurrUp(k) = Min(OptProport(k) + 2 ^ (-j - 1), AbsUp(k))
       Next k
    Next j

  ' CALCULATE VALUES TO SEND AS OUTPUT
    OptMean = 0
    BlendVar = 0
    For k = 1 To NumSources
       If OptProport(k) < 0 Then
          OptProport(k) = 0
       End If
```

```
            OptMean = OptMean + (Mu(k) * OptProport(k))
            BlendVar = BlendVar + ((OptProport(k) ^ 2) * Var(k))
        Next k
        OptStandDev = Sqr(BlendVar)
        OptMax = OptMean + Z * OptStandDev

End Sub
```

APPENDIX 11

Code for "What-if" Analysis for 2 Sources

```
Public Sub WIComputeTwo(NumSources As Integer, MaxSafeVal As Single,
          BMean As Single, BStandDev As Single, WIProb As Single)

  ' *** THIS PROCEDURE IS THE CODE FOR CALCULATING
  '     A "WHAT-IF" ANALYSIS FOR 2 SOURCES ***


  ' Declare Local Variables
   Dim i As Integer
   Dim Ki(1 To 2) As Single              ' Blend Proportions
   Dim Mu(1 To 2) As Single              ' Source Mean
   Dim Var(1 To 2) As Single             ' Source Variation
   Dim Z As Single                       ' Determines Probability
   Dim NumericErr As Boolean             ' Detects errors in Source Data
   Dim ProportErr As Boolean             ' Detects error in Blend Proportion
   Dim MeanErr As Boolean                ' Detects error in Source Mean
   Dim VarErr As Boolean                 ' Detects error in Source Variation
   Dim ProportSum As Single              ' Sums the Blend Proportions
   Dim BVariance As Single               ' Blend Variance

  ' Initialize Variables
   ProportErr = False
   MeanErr = False
   VarErr = False
   ProportSum = 0

   Ki(1) = frmSourceOne!txtSourceOneKi.Text
   Ki(2) = frmSourceTwo!txtSourceTwoKi.Text

   Mu(1) = frmSourceOne!txtSourceOneMean.Text
   Mu(2) = frmSourceTwo!txtSourceTwoMean.Text

   If (frmSourceOne!optSourceOneSD.Value = True) Then
     Var(1) = (frmSourceOne!txtSourceOneVar.Text) ^ 2
   Else
     Var(1) = (frmSourceOne!txtSourceOneVar.Text)
   End If
   If (frmSourceTwo!optSourceTwoSD.Value = True) Then
     Var(2) = (frmSourceTwo!txtSourceTwoVar.Text) ^ 2
   Else
     Var(2) = (frmSourceTwo!txtSourceTwoVar.Text)
   End If

  ' Error-Check Variables (Probs sum to 1; non-negative values)
   For i = 1 To 2
     If Ki(i) < 0 Then
        ProportErr = True
     End If
     If Mu(i) < 0 Then
        MeanErr = True
     End If
     If Var(i) < 0 Then
        VarErr = True
     End If
     ProportSum = ProportSum + Ki(i)
   Next i

  ' If Error Exists, Display Appropriate Message and Exit Subroutine
   If ((ProportErr) Or (ProportSum <> 1)) Then
     MsgBox "Proportions must be non-negative and sum to 1. Try again.", 48, "C
annot Compute"
     Exit Sub
   End If
   If MeanErr = True Then
     MsgBox "Source means must be non-negative. Try again.", 48, "Cannot Comput
e"
     Exit Sub
```

```
    End If
    If VarErr = True Then
      MsgBox "Source variations must be non-negative. Try again.", 48, "Cannot C
ompute"
      Exit Sub
    End If

   ' Calculate Values
    For i = 1 To 2
      BMean = BMean + Ki(i) * Mu(i)
      BVariance = BVariance + (Ki(i) ^ 2) * (Var(i))
    Next i
    BStandDev = Sqr(BVariance)
    If BStandDev = 0 Then
      If BMean > MaxSafeVal Then
         WIProb = 1
      Else
         WIProb = 0
      End If
    Else
      Z = (MaxSafeVal - BMean) / BStandDev
      Call SNormInv(Z, WIProb)
    End If

End Sub
```

APPENDIX 12

Validation Data

Validation Data

n = 3

$\mu_1 = 50$	$\sigma_1^2 = 4$	$b_1 = 1.00$
$\mu_2 = 40$	$\sigma_2^2 = 36$	$b_2 = 1.00$
$\mu_3 = 45$	$\sigma_3^2 = 12$	$b_3 = 1.00$

$M_p^* = 52.16074$

experimental runs
1) 52.16074
2) 52.16074
3) 52.16074
4) 52.16074
5) 52.16074

% error = (52.16074 - 52.16074) / 52.16074
= 0.00%

n = 7

$\mu_1 = 50$	$\sigma_1^2 = 4$	$b_1 = 1.00$
$\mu_2 = 40$	$\sigma_2^2 = 36$	$b_2 = 1.00$
$\mu_3 = 45$	$\sigma_3^2 = 12$	$b_3 = 0.09$
$\mu_4 = 51.5$	$\sigma_4^2 = 0.01$	$b_4 = 1.00$
$\mu_5 = 47$	$\sigma_5^2 = 20$	$b_5 = 1.00$
$\mu_6 = 51$	$\sigma_6^2 = 1$	$b_6 = 1.00$
$\mu_7 = 42$	$\sigma_7^2 = 27$	$b_7 = 1.00$

$M_p^* = 51.17793$

experimental runs
1) 51.19986
2) 51.28685
3) 51.23554
4) 51.31067
5) 51.28316

% error = (51.263216 - 51.17793) / 51.17793
= 0.17%

n = 10

$\mu_1 = 50$	$\sigma_1^2 = 4$	$b_1 = 1.00$
$\mu_2 = 40$	$\sigma_2^2 = 36$	$b_2 = 1.00$
$\mu_3 = 45$	$\sigma_3^2 = 12$	$b_3 = 1.00$
$\mu_4 = 43$	$\sigma_4^2 = 17$	$b_4 = 1.00$
$\mu_5 = 47$	$\sigma_5^2 = 21$	$b_5 = 1.00$
$\mu_6 = 41$	$\sigma_6^2 = 14$	$b_6 = 0.03$
$\mu_7 = 42$	$\sigma_7^2 = 33$	$b_7 = 1.00$
$\mu_8 = 49$	$\sigma_8^2 = 7$	$b_8 = 1.00$
$\mu_9 = 48$	$\sigma_9^2 = 11$	$b_9 = 1.00$
$\mu_{10} = 46$	$\sigma_{10}^2 = 6$	$b_{10} = 1.00$

$M_p^* = 48.92204$

experimental runs
1) 48.92244
2) 49.30835
3) 49.02160
4) 48.92251
5) 48.93250

% error = (49.02148 - 48.92204) / 48.92204
= 0.20%